

# Hybrid Mobile Applications - ICD0018

TalTech IT College, Andres Käver, 2021-2022, Fall semester

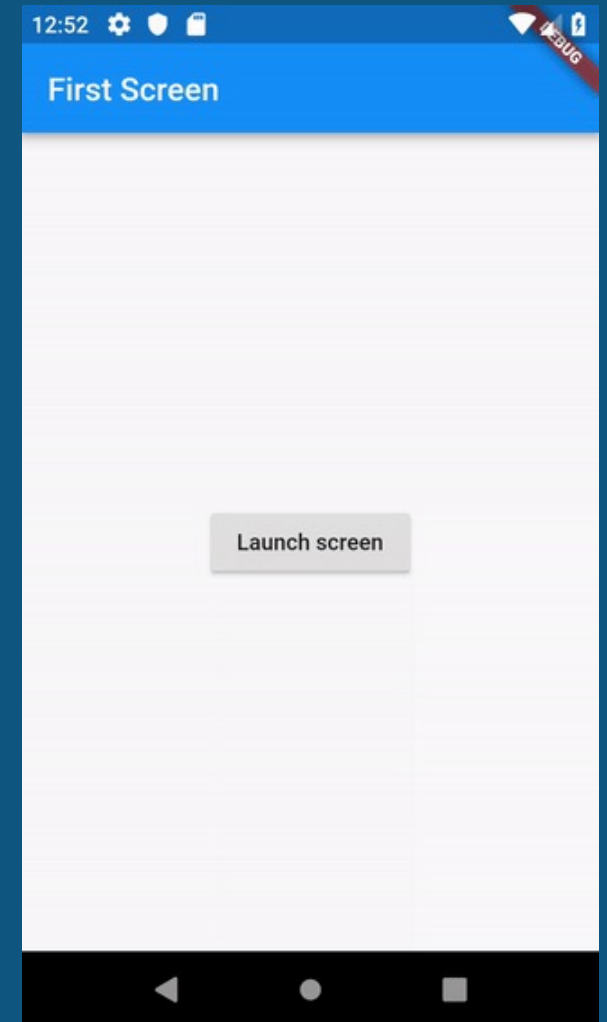
Email: [andres.kaver@taltech.ee](mailto:andres.kaver@taltech.ee)



# Flutter – Navigation & Routing

2

- ▶ Screens and pages are called Routes
  - ▶ Android – Route is Activity
  - ▶ iOS – Route is ViewController
  - ▶ Flutter – route is just a widget



# Flutter - Navigator

3

- ▶ `Navigator.push(context, route)` – to move to next route
- ▶ `Navigator.pop(context)` – to move back to previous route
- ▶ Route?
  - ▶ Create with `MaterialPageRoute(builder: (context) => SecondRoute())`

# Flutter – Push route

4

- ▶ Navigate to next view

```
class FirstRoute extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('First Route'),  
      ),  
      body: Center(  
        child: RaisedButton(  
          child: Text('Open route'),  
          onPressed: () {  
            Navigator.push(  
              context,  
              MaterialPageRoute(builder: (context) => SecondRoute()),  
            );  
          },  
        ),  
      ),  
    );  
  }  
}
```

# Flutter – Pop route

5

- ▶ Navigate back to previous view

```
class SecondRoute extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Second Route"),  
      ),  
      body: Center(  
        child: RaisedButton(  
          onPressed: () {  
            Navigator.pop(context);  
          },  
          child: Text('Go back!'),  
        ),  
      ),  
    );  
  }  
}
```

# Flutter – Named routes

6

- ▶ To avoid code duplication, when you need to navigate to same screen from several places
- ▶ `Navigator.pushNamed`
- ▶ `Navigator.pop`
- ▶ When defining `initialRoute` – don't define `home` property!

```
void main() {  
  runApp(MaterialApp(  
    title: 'Named Routes Demo',  
    initialRoute: '/',  
    routes: {  
      '/': (context) => FirstScreen(),  
      '/second': (context) => SecondScreen(),  
    },  
  ));  
}
```

```
onPressed: () {  
  // Navigate to the second screen using a named route.  
  Navigator.pushNamed(context, '/second');  
},
```

# Flutter – Passing data to/from route

7

- ▶ Pass data the regular way – everything is just widget!
- ▶ To receive data
  - ▶ `Navigator.push` returns `Future`, that completes after pop in new screen
- ▶ To send data back
  - ▶ `Navigator.pop(context, 'Option A was chosen!');`

# Flutter - Passing data to/from route

8

- ▶ Async/Await pattern – same as in C# async

```
_navigateAndDisplaySelection(BuildContext context) async {  
  final result = await Navigator.push(  
    context,  
    MaterialPageRoute(  
      builder: (context) => SelectionScreen(  
        textToShow: 'Choose one!',  
      )),  
  );  
  
  Scaffold.of(context)  
    ..removeCurrentSnackBar()  
    ..showSnackBar(SnackBar(content: Text("$result")));  
}
```



# Flutter - Tabs

- ▶ DefaultTabController
  - ▶ TabBar
  - ▶ TabBarView
- ▶ Order must match!

```
class TabBarDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DefaultTabController(
        length: 3,
        child: Scaffold(
          appBar: AppBar(
            bottom: TabBar(
              tabs: [
                Tab(icon: Icon(Icons.directions_car)),
                Tab(icon: Icon(Icons.directions_transit)),
                Tab(icon: Icon(Icons.directions_bike)),
              ],
            ),
          title: Text('Tabs Demo'),
        ),
        body: TabBarView(
          children: [
            Icon(Icons.directions_car),
            Icon(Icons.directions_transit),
            Icon(Icons.directions_bike),
          ],
        ),
      ),
    );
  }
}
```

# Flutter - Orientation

10

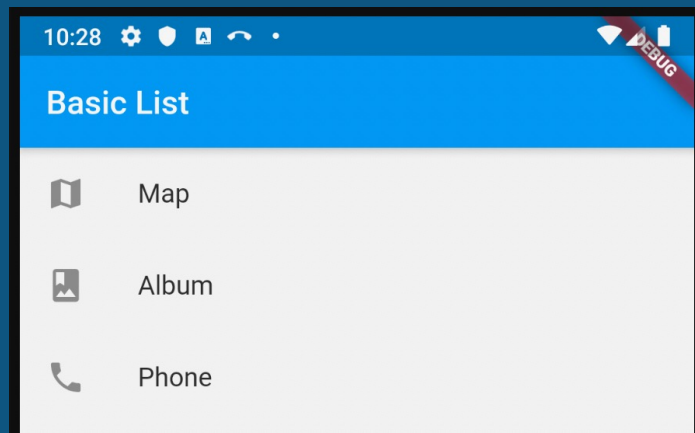
- Use OrientationBuilder

```
class OrientationController extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return OrientationBuilder(  
      builder: (context, orientation) {  
        return orientation == Orientation.landscape  
          ? LandscapeLayout()  
          : PortraitLayout();  
      },  
    );  
  }  
}
```

# Flutter - List

11

- ▶ Lists are fundamental mobile apps development components (Facebook app is list, etc)
- ▶ Flutter has ListView widget
- ▶ ListTile gives every row some structure
- ▶ Or create custom widget



```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    final title = 'Basic List';  
  
    return MaterialApp(  
      title: title,  
      home: Scaffold(  
        appBar: AppBar(  
          title: Text(title),  
        ),  
        body: ListView(  
          children: <Widget>[  
            ListTile(  
              leading: Icon(Icons.map),  
              title: Text('Map'),  
            ),  
          ],  
        ),  
      ),  
    );  
  }  
}
```

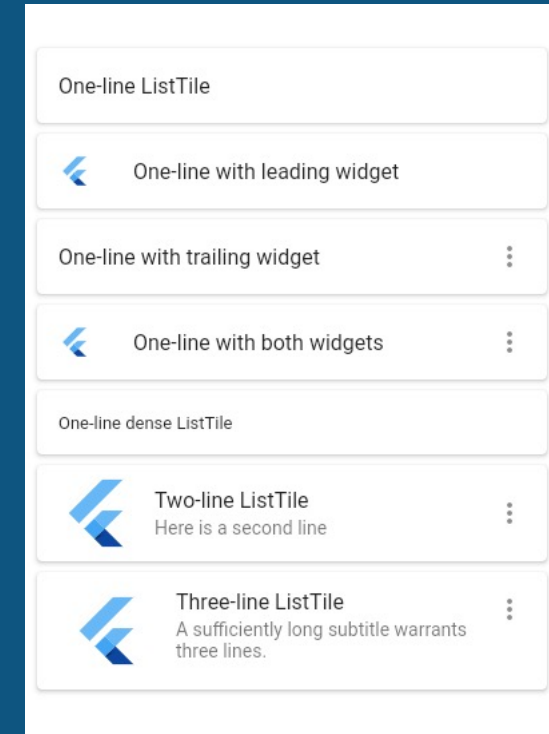
# Flutter – Card/ListTile

12

```
children: <Widget>[  
  Card(  
    child: ListTile(  
      leading: FlutterLogo(size: 56.0),  
      title: Text('Two-line ListTile'),  
      subtitle: Text('Here is a second line'),  
      trailing: Icon(Icons.more_vert),  
    ),  
  ),  
],
```

## ► Properties

contentPadding, dense, enabled, isThreeLine, leading, onLongPress, onTap, selected, subtitle, title, trailing



# Flutter – Long lists

13

- ▶ If lists are really long (or dynamical)
- ▶ Use `ListView.builder` constructor
- ▶ Generate list items in **itemBuilder**

```
body: ListView.builder(  
    itemCount: items.length,  
    itemBuilder: (context, index) {  
        return ListTile(  
            title: Text('Item: ${index}'),  
        );  
    },  
),
```

# Flutter - Images

14

- ▶ Image class, several constructors
  - ▶ new Image, for obtaining an image from an ImageProvider.
  - ▶ new Image.asset, for obtaining an image from an AssetBundle using a key.
  - ▶ new Image.network, for obtaining an image from a URL.
  - ▶ new Image.file, for obtaining an image from a File.
  - ▶ new Image.memory, for obtaining an image from a Uint8List.
- ▶ Image.network('https://flutter.github.io/assets-for-api-docs/assets/widgets/owl-2.jpg')

# Flutter - Image, fadein

15

- ▶ Pubspec: transparent\_image: ^1.0.0

```
body: Stack(  
  children: <Widget>[  
    Center(child: CircularProgressIndicator()),  
    Center(  
      child: FadeInImage.memoryNetwork(  
        placeholder: kTransparentImage,  
        image: 'https://picsum.photos/250?image=9',  
      ),  
    ],  
),
```

# Flutter - SnackBar

16

- ▶ Create a Scaffold
  - ▶ Ensures, that widgets don't overlap
- ▶ Create a SnackBar
- ▶ Display SnackBar, using Scaffold

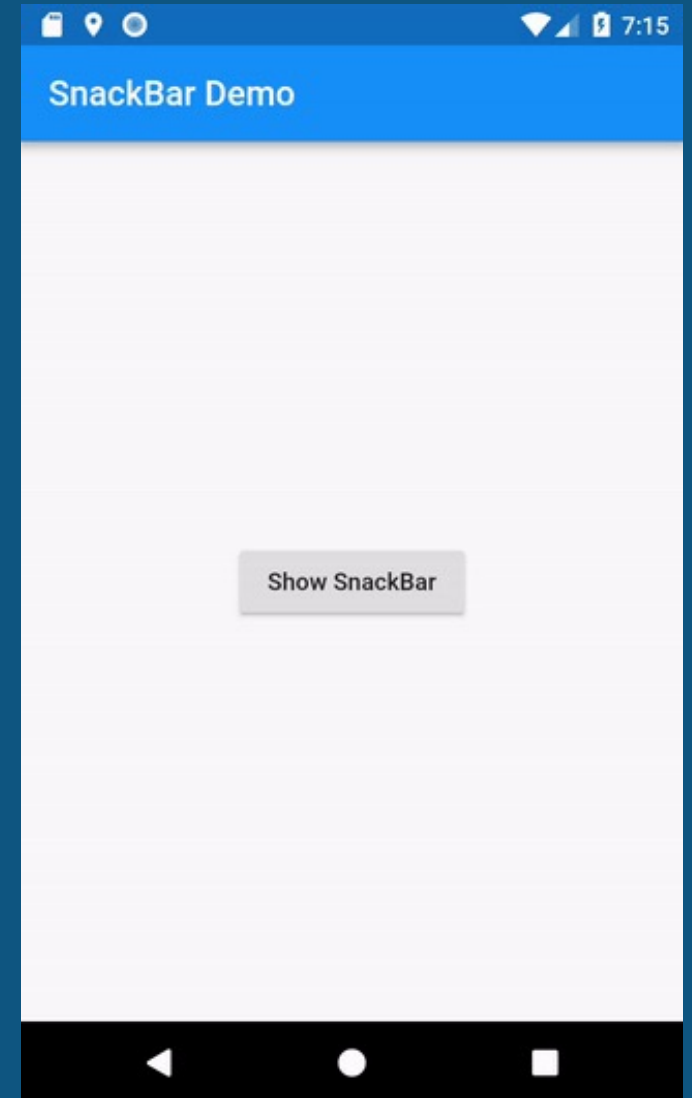
```
class SnackBarDemo extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'SnackBar Demo',  
      home: Scaffold(  
        appBar: AppBar(  
          title: Text('SnackBar Demo'),  
        ),  
        body: SnackBarPage(),  
      ),  
    );  
  }  
}
```



# Flutter - Snackbar

17

```
class SnackbarPage extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Center(  
      child: RaisedButton(  
        onPressed: () {  
          final snackBar = SnackBar(  
            content: Text('Yay! A Snackbar!'),  
            action: SnackBarAction(  
              label: 'Undo',  
              onPressed: () {},  
            ),  
          );  
          Scaffold.of(context).showSnackBar(snackBar);  
        },  
        child: Text('Show Snackbar'),  
      ),  
    );  
  }  
}
```



- ▶ Users need to interact with the app and sometimes built-in functionality is not enough
- ▶ GestureDetector

```
class MyButton extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return GestureDetector(  
      onTap: () {  
        final snackBar = SnackBar(content: Text("Tap"));  
        Scaffold.of(context).showSnackBar(snackBar);  
      },  
      // The custom button  
      child: Container(  
        padding: EdgeInsets.all(12.0),  
        decoration: BoxDecoration(  
          color: Theme.of(context).buttonColor,  
          borderRadius: BorderRadius.circular(8.0),  
        ),  
        child: Text('My Button'),  
      ),  
    );  
  }  
}
```

# Flutter - GestureDetector

19

- ▶ onDoubleTap
- ▶ onForcePressEnd
- ▶ onForcePressPeak
- ▶ onForcePressStart
- ▶ onForcePressUpdate
- ▶ onHorizontalDragCancel
- ▶ onHorizontalDragDown
- ▶ onHorizontalDragEnd
- ▶ onHorizontalDragStart
- ▶ onHorizontalDragUpdate
- ▶ onLongPress
- ▶ onLongPressEnd
- ▶ onLongPressMoveUpdate
- ▶ onLongPressStart
- ▶ onLongPressUp
- ▶ onPanCancel
- ▶ onPanDown
- ▶ onPanEnd
- ▶ onPanStart
- ▶ onPanUpdate
- ▶ onScaleEnd
- ▶ onScaleStart
- ▶ onScaleUpdate
- ▶ onSecondaryTapCancel
- ▶ onSecondaryTapDown
- ▶ onSecondaryTapUp
- ▶ onTap
- ▶ onTapCancel
- ▶ onTapDown
- ▶ onTapUp
- ▶ onVerticalDragCancel
- ▶ onVerticalDragDown
- ▶ onVerticalDragEnd
- ▶ onVerticalDragStart
- ▶ onVerticalDragUpdate

# Flutter – Fetch data

20

- ▶ Package: http  
dependencies: **http: ^0.12.2**  
**import** 'package:http/http.dart';
- ▶ Get() returns an Future containing Response (await/async pattern)
- ▶ Convert response to Dart object
- ▶ Factory constructors  
Use the factory keyword when implementing a constructor that doesn't always create a new instance of its class. For example, a factory constructor might return an instance from a cache, or it might return an instance of a subtype.  
No access to 'this'!

```
Future<http.Response> fetchInfo () {  
    return http.get('http://dad.akaver...');  
}
```

```
class SongInfo {  
    final String artist;  
    final String title;  
    SongInfo({this.artist, this.title});  
    factory SongInfo.fromJson(Map<String, dynamic> json){  
        var songHistoryList = json['SongHistoryList'];  
        var song = songHistoryList[0];  
        return SongInfo(  
            artist: song['Artist'],  
            title: song['Title']  
        );  
    }  
}
```

# Flutter – Fetch data

21

- ▶ Convert json to object

```
Future<SongInfo> fetchInfo() async {  
  final response =  
    await http.get('http://dad.akaver.com/api/SongTitles/SP');  
  
  if (response.statusCode == 200) {  
    return SongInfo.fromJson(json.decode(response.body));  
  } else {  
    throw Exception('Failed to load data');  
  }  
}
```

# Flutter – Fetch data

22

- ▶ Fetch the data!
- ▶ Create future in initState
  - ▶ Build method gets called often (UI redraw) – slowdown if requests where to happen every time

```
class MyApp extends StatefulWidget {  
  MyApp({Key key}) : super(key: key);  
  
  @override  
  _MyAppState createState() => _MyAppState();  
}  
  
class _MyAppState extends State<MyApp> {  
  Future<SongInfo> info;  
  
  @override  
  void initState() {  
    super.initState();  
    info = fetchInfo();  
  }  
}
```

# Flutter – Fetch data / display

23

- ▶ `FutureBuilder<YourFutureDataClass>`
  - ▶ Builder to work with async data sources
  - ▶ Future – async data source to work with
  - ▶ Builder – what to render during
    - ▶ Loading
    - ▶ Success
    - ▶ Failure

```
body: Center(  
  child: FutureBuilder<SongInfo>(  
    future: info,  
    builder: (context, snapshot) {  
      if (snapshot.hasData) {  
        return Text(  
          snapshot.data.artist + ' - ' +  
          snapshot.data.title);  
      } else if (snapshot.hasError) {  
        return Text("${snapshot.error}");  
      }  
      return CircularProgressIndicator();  
    },  
  ),  
)
```

# Flutter - Timer

24

- ▶ Periodic callback

```
Timer _timer;  
int _start = 10;  
  
void startTimer() {  
  const oneSec = const Duration(seconds: 1);  
  _timer = new Timer.periodic(  
    oneSec,  
    (Timer timer) => setState(  
      () {  
        if (_start < 1) {  
          timer.cancel();  
          info = fetchInfo();  
        } else {  
          _start = _start - 1;  
        }  
      },  
    ),  
  );  
}
```



- ▶ Still to come... maybe....
- ▶ Persistence
  - ▶ Networking
    - ▶ Tokens/Auth, Json parsing in background, WebSockets/SignalR
  - ▶ Working with SQLite
  - ▶ Files, Key-Value data
- ▶ Cupertino vs Material
- ▶ Background services (Isolate), Media, notifications, push, sensors
- ▶ Plugins – interfacing with native API's
- ▶ i18n, accessibility
- ▶ Animations, Assets

# Flutter -

26

# Flutter -

27